DT002G, Applied Computer Engineering

IaC Template Suite for Credential-Free Workload Identity Provisioning

Software Requirements Specification

Mid Sweden University

DSV Östersund

Examinator	Sergio Rico	sergio.rico@miun.se
Supervisor	Sergio Rico	sergio.rico@miun.se
Author 1	Albin Rönnkvist	alrn1700@student.miun.se
Author 2	Piran Amedi	roam2200@student.miun.se
Programme	Software Engineering, 180hp	
Course	DT002G, Applied Computer Engineering	
Field of study	Computer Engineering	
semester, year	VT, 2025	

Table of Contents

1 Introduction

As organizations increasingly transition to cloud and hybrid environments, Identity and Access Management (IAM)[1] for workloads—including applications, services, and automated processes—has become a critical component of securing cloud resources [1, 2]. Traditionally, IAM has been managed manually through cloud provider portals or ad hoc scripts, often leading to configuration drift, limited visibility, and challenges in auditing [3, 4]. Moreover, workload IAM implementations traditionally rely on authentication using long-lived static credentials such as secrets and certificates [3, 4, 5]. These credentials require secure storage, regular rotation, and manual management. As organizations scale their workloads, these issues become more pronounced, leading to operational complexity and fragile security posture [5]. To address these challenges, this project introduces an Infrastructure as Code (IaC)[2]-driven template suite that acts as the source of truth for managing credential-free[3] workload identities[4] and their access to cloud resources. The proposed approach provisions workload identities and role-based access to Azure cloud resources using Managed Identities (MIs)[5] for internal cloud workloads and Federated Identity Credentials (FICs)[6] for external workloads. This approach prevents configuration drift, enhances visibility, and allows workloads to authenticate securely without managing long-lived static credentials. By providing an installable, customizable IaC template suite, organizations can quickly adopt a secure and scalable credential-free workload IAM framework without needing to build one from scratch.

1.1 Problem Statement & Purpose

Managing IAM for workloads in cloud and hybrid environments at scale poses significant challenges when relying on manual configurations and long-lived credentials. These traditional methods lead to operational complexity, security risks, and difficulty in maintaining visibility and control, as described in the Introduction. To address these issues, a scalable, automated, and credential-free workload IAM framework is needed. This project aims to develop an IaC-driven template suite that automates workload identity provisioning and access management. By leveraging version-controlled infrastructure code, it seeks to enhance visibility, prevent configuration drift, and simplify maintenance. Additionally, by replacing long-lived credentials with FICs and MIs, it aims to strengthen security and reduce maintenance efforts. The goal is to provide organizations with a secure, scalable, and easily deployable workload IAM framework that minimizes operational overhead, development time, and costs.

[1] https://learn.microsoft.com/entra/fundamentals/introduction-identity-access-management
[2] https://www.redhat.com/topics/automation/what-is-infrastructure-as-code-iac
[3] Credential-free authentication eliminates the need to manually manage secrets, certificates, and keys for secure service communication.
[4] https://learn.microsoft.com/entra/workload-id/workload-identities-overview
[5] https://learn.microsoft.com/entra/identity/managed-identities-azure-resources/
[6] https://learn.microsoft.com/graph/api/resources/federatedidentitycredentials-overview

1.2 Scope

This project focuses on developing a suite of integrated IaC templates using Pulumi and C# .NET, enabling organizations to provision workload identities in Microsoft Entra ID with role-based access control (RBAC)[7] for Azure resources. The solution will incorporate Microsoft Entra ID Applications[8] with FICs, allowing external workloads to securely authenticate and access Azure services. Additionally, it will leverage MIs to facilitate secure access for internal Azure workloads.

1.3 Completeness Criteria

The project will be considered complete when a fully functional suite of Pulumi- and C#-based IaC templates is available for installation, allowing organizations to set up provisioning of credential-free workload identities with least privilege[9] RBAC assignments for accessing Azure resources. The solution must fulfill the following requirements:

- **Secure External Access:** External workloads, such as CI/CD pipelines, must use FICs for authentication to Azure resources.
- **Secure Internal Access:** Internal workloads running on Azure must use MIs for authentication to Azure resources.
- **Least Privilege Enforcement:** All workload identities must be assigned least privilege Azure roles, ensuring they have access only to the necessary resources.
- **Automated and Customizable IaC Templates:** The IaC templates must be automated, reusable, and customizable, allowing organizations to tailor them to their specific requirements.
- **Comprehensive Documentation:** Detailed documentation must be provided, covering installation steps, customization options, and best practices to ensure seamless adoption.
- **Thorough Testing and Validation:** The solution must undergo extensive testing and validation to ensure real-world usability.

[7] https://learn.microsoft.com/azure/role-based-access-control/overview
[8] https://learn.microsoft.com/en-us/entra/identity-platform/app-objects-and-service-principals
[9] https://learn.microsoft.com/entra/identity-platform/secure-least-privileged-access

2 Overall Description

This section provides an overview of the Template Suite product, a set of four integrated templates designed to facilitate the provisioning of credential-free workload identities and secure access management within an organization’s cloud infrastructure. It begins by introducing the system’s features, detailing their functionality and purpose. Thereafter, the design and implementation constraints are outlined, describing how limitations influence the final product. The assumptions and dependencies required for the system’s operation are then examined, for clarity on its prerequisites. Finally, the user classes and their roles within the system are introduced, describing how different participants interact with the solution.

2.1 Product Features

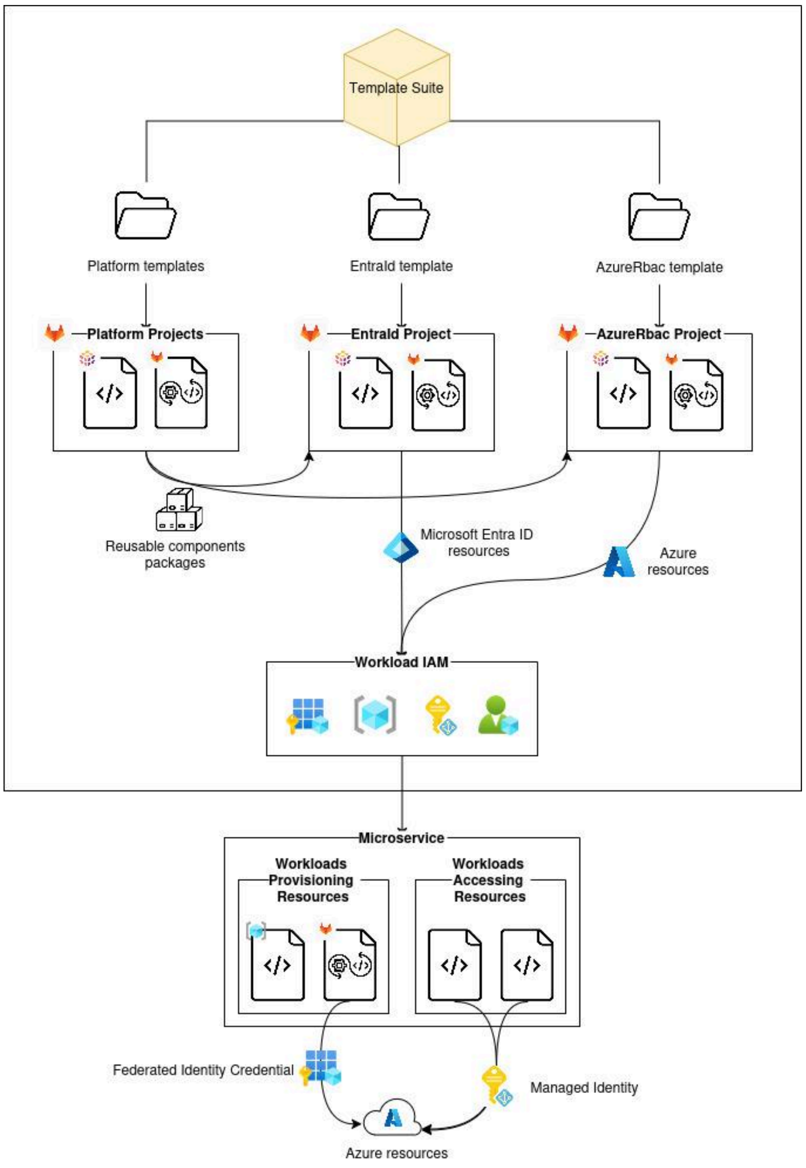

Figure 1: Product Overview

The template suite package consists of four integrated templates that streamline infrastructure setup:

1. **PlatformPulumi**: Provides reusable Pulumi components and standardized conventions.
2. **PlatformPulumiAzure**: Extends *PlatformPulumi* with Azure-specific components.
3. **EntraId**: Creates Microsoft Entra ID Applications with FICs for secure authentication in external workloads.
4. **AzureRbac**: Provisions Azure Resource Groups, restricting microservices to designated environments, creates MIs for internal workload authentication, and assigns RBAC roles for controlled access to Azure resources.

Together, these templates enable the following capabilities:

- Automates provisioning of FICs and MIs, enabling secure, credential-free authentication to Azure resources for internal and external workloads.
- Manages Role-Based Access Control (RBAC) for workloads, ensuring controlled access to Azure resources.
- Ensures each microservice and its workloads are contained within an isolated Azure Resource Group, with assigned identities and roles, restricting access to only its designated resources.
- Provides modular IaC, with Pulumi Stack References[10] for sharing resource information between projects, while ensuring access is restricted to authorized identities.

2.2 Design and Implementation Constraints

Due to time constraints, the templates will prioritize compatibility with a specific set of tools, such as a single cloud provider and DevOps platform. As a result, integration with multiple similar tools (e.g., other cloud providers, alternative DevOps services, or language-specific implementations like C#) will be limited.

In addition to lack of tooling support, certain functionalities may not be fully supported in Pulumi, potentially requiring workarounds or alternative implementation strategies. In particular, Federated Identity Credentials is a relatively new feature and may lack support for advanced capabilities. For instance, Flexible Federated Identity Credentials[11], an enhanced version of the feature, is still in preview and may undergo changes that could impact long-term stability and compatibility.

2.3 Assumptions and Dependencies

Maintainers of this project are expected to have experience with cloud providers and IAM solutions, preferably Azure and Microsoft Entra ID, as well as C# .NET expertise for working with the generated projects. However, these skills are not strictly required for the solution's functionality.

The project depends on the organization operating within the Microsoft ecosystem, utilizing Microsoft Entra ID and Azure for IAM and cloud resources. The setup requires an Azure tenant with two subscriptions (one for development and one for production) and a user with Global Administrator privileges. Additionally, the solution depends on Pulumi as the IaC tool, requiring Pulumi Cloud to be configured, and the Pulumi CLI to be installed locally. Furthermore, the resulting infrastructure code must be stored in GitLab repositories within

[10] https://www.pulumi.com/tutorials/building-with-pulumi/stack-references/
[11] https://learn.microsoft.com/entra/workload-id/workload-identities-flexible-federated-identity-credentials

the same group, as the generated GitLab CI/CD pipelines rely on it for automated deployment and identity management. This also requires setting up access to Pulumi in GitLab. Since the templates are distributed as .NET templates via a NuGet package, the .NET SDK is required for installation and usage. Lastly, workloads within the organization must support FICs or MIs for authentication.

2.4 User Classes and Characteristics

The system is designed for different user roles, each with specific responsibilities related to IAM and Azure resource provisioning.

2.4.1 Global Administrator

The Global Administrator is responsible for the initial setup of the system, including configuring the Azure tenant, Pulumi Cloud, GitLab and installing the templates.

2.4.2 IAM Administrators

IAM administrators are assigned two distinct areas of responsibility:

- **IAM Entra ID Administrators** – Manage Microsoft Entra ID applications, including the creation of FICs and assigning directory roles as needed.
- **IAM Azure RBAC Administrators** – Oversee MIs and access management by defining Azure roles and permissions for all workloads accessing Azure resources.

2.4.3 Operations & Developer Teams

The Operations and Developer teams each have distinct roles in managing and utilizing cloud resources:

- **Operations Team** – Responsible for provisioning Azure resources for microservices and creating CI/CD pipelines. They utilize the provisioned FICs to securely access Azure resources within CI/CD pipelines.
- **Developer Team** – Focused on building microservice applications, such as APIs and background jobs. They use MIs to securely access Azure resources within their workloads, such as retrieving secrets from Azure Key Vault in a running API container.

3 Use Cases

The Use Cases define the interactions between Administrators, Operations and Developer teams, and other actors to establish, manage, and utilize the IAM infrastructure. These use cases outline the end-to-end lifecycle of the system, covering installation, configuration, management, and utilization.

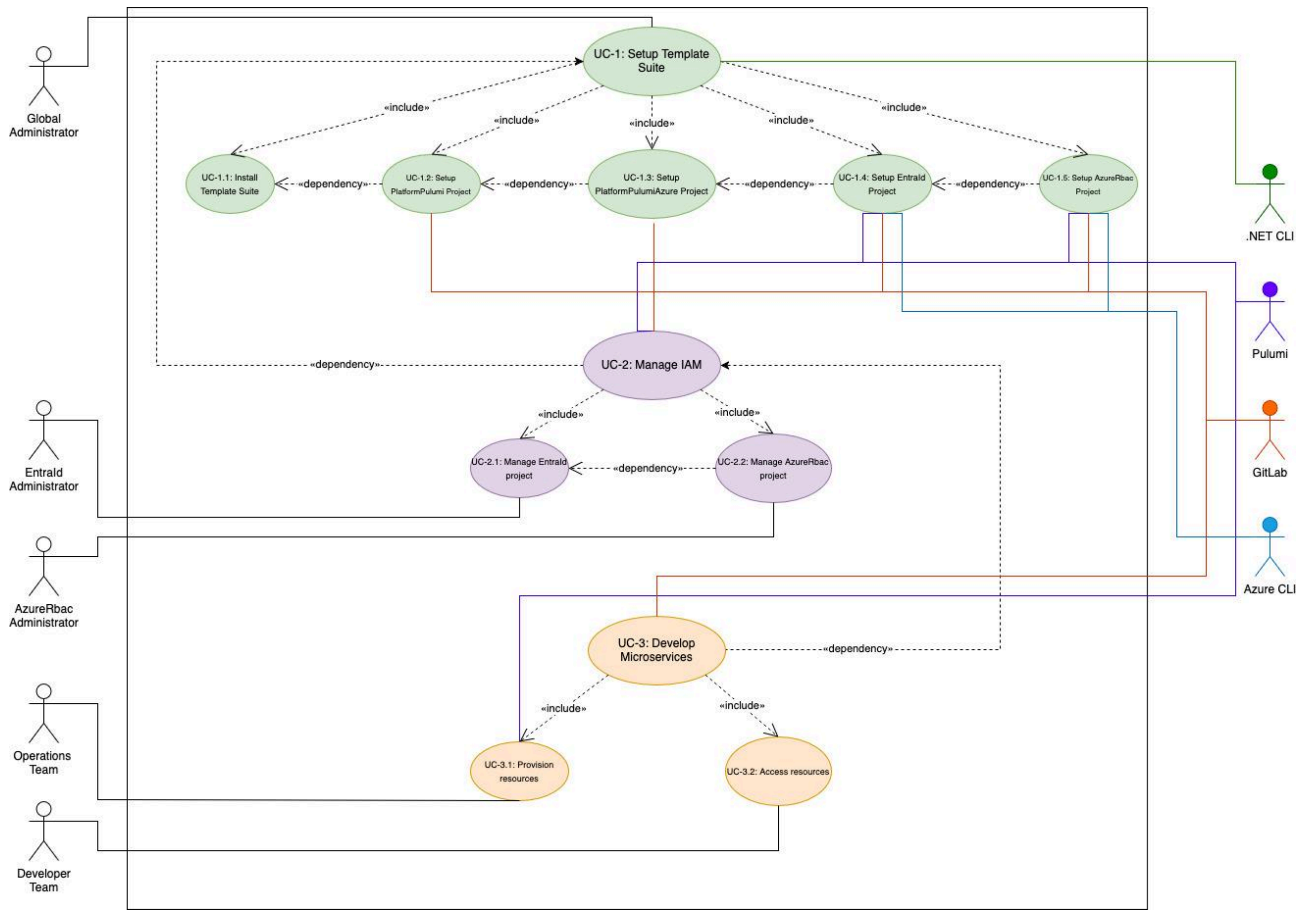

Figure 2: Use case diagram

The Global Administrator is responsible for setting up the system, as outlined in Appendix A. This includes installing the necessary templates (Appendix B) and ensuring the availability of common NuGet packages, such as *PlatformPulumi* (Appendix C) and *PlatformPulumiAzure* (Appendix D). Additionally, they oversee the initial provisioning of *IaC* projects, including *EntraId* (Appendix E) and *AzureRbac* (Appendix F), to establish the foundational infrastructure. Once the foundational infrastructure is in place, the IAM (EntraId and AzureRbac) Administrators manage IAM within their respective domains to ensure secure authentication and controlled access for workloads (Appendix G). EntraId Administrators handle identity and role assignments within *Microsoft Entra ID* (Appendix H), while AzureRbac Administrators manage access to Azure resources through *Role-Based Access Control (RBAC) policies* (Appendix I). With identity and access controls in place, the Operations and Developer teams utilize the granted identities and permissions within the established IaC framework to provision and access resources (Appendix J). The Operations team uses assigned workload identities to provision resources within authorized resource groups (Appendix K). Meanwhile, the Developer team leverages the provided MI to securely access resources within the group, enabling interaction between their applications and Azure services (Appendix L).

4 Functional Requirements

The following Functional Requirements define the capabilities necessary to support the initial system setup, IAM administration, and resource provisioning workflows described in Section 3. These requirements ensure that the Global Administrator can efficiently install and configure the system, IAM Administrators can manage secure access to Azure resources, and the Operations and Developer teams can gain least privilege access to provision and use resources within the established infrastructure. Each requirement is structured with a unique ID, title, description, rationale, and dependencies.

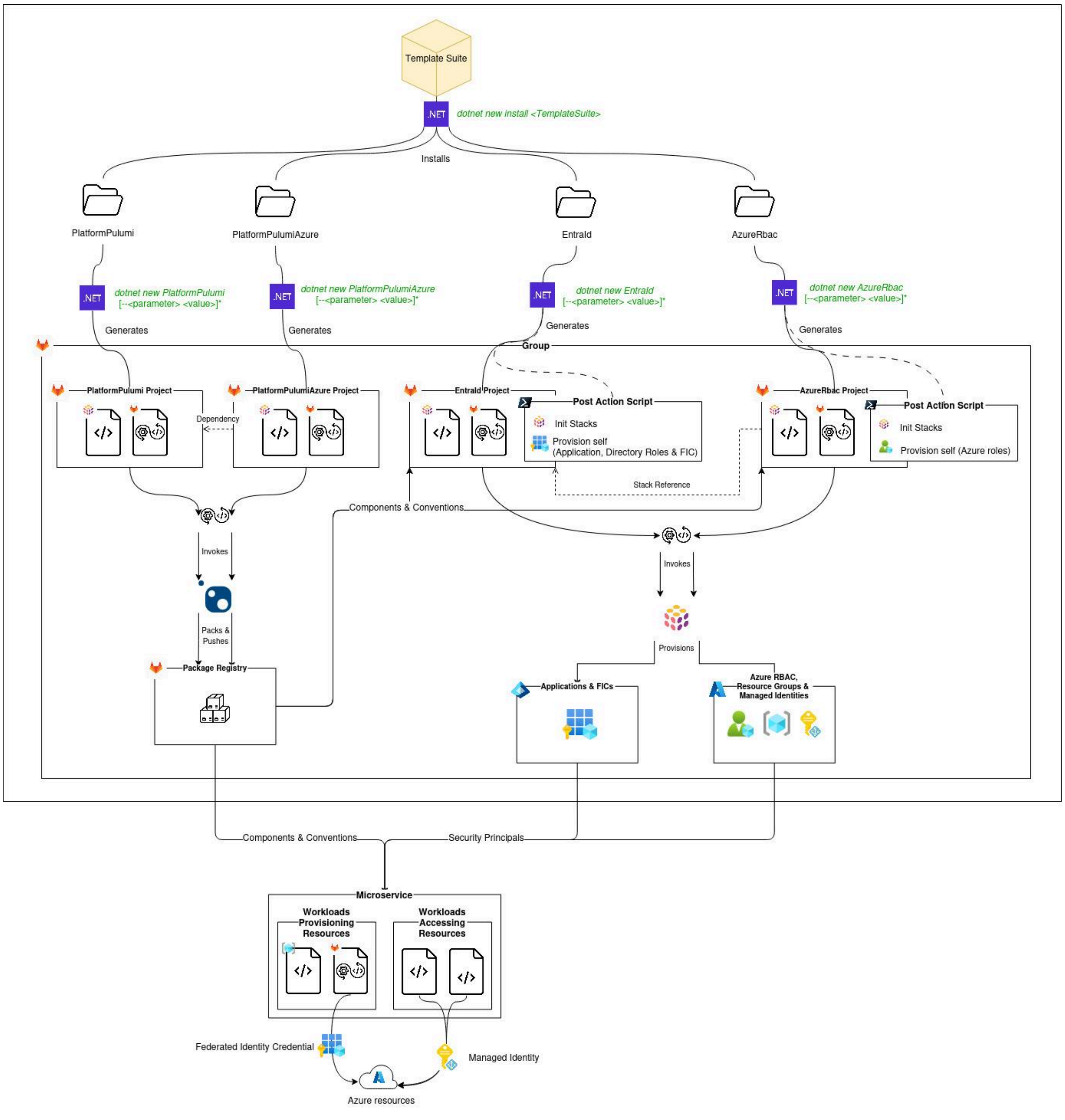
Template Suite
.NET
dotnet new install <TemplateSuite>
Installs
PlatformPulumi
PlatformPulumiAzure
EntraId
AzureRbac
.NET
dotnet new PlatformPulumi [--<parameter> <value>]*
.NET
dotnet new PlatformPulumiAzure [--<parameter> <value>]*
.NET
dotnet new EntraId [--<parameter> <value>]*
.NET
dotnet new AzureRbac [--<parameter> <value>]*
Generates
Generates
Generates
Generates
Group
PlatformPulumi Project
PlatformPulumiAzure Project
Dependency
EntraId Project
Post Action Script
Init Stacks
Provision self (Application, Directory Roles & FIC)
AzureRbac Project
Post Action Script
Init Stacks
Provision self (Azure roles)
Stack Reference
Components & Conventions
Invokes
Packs & Pushes
Package Registry
Invokes
Provisions
Applications & FICs
Azure RBAC, Resource Groups & Managed Identities
Components & Conventions
Security Principals
Microservice
Workloads Provisioning Resources
Workloads Accessing Resources
Federated Identity Credential
Managed Identity
Azure resources

Figure 3: Functional requirements diagram

4.1 Global Administrator

ID	FR1
TITLE	Install Template Suite

DESC	The template suite must be installable as one package using .NET SDK.
RAT	Ensures a streamlined installation process, reducing manual intervention.
DEP	None

ID	FR2
TITLE	Generate Projects from Templates with Custom Parameters
DESC	The suite's templates must generate projects with customized values based on supported parameters.
RAT	Streamlines project setup while maintaining adaptability to different requirements.
DEP	FR1

ID	FR3
TITLE	Publish NuGet Packages for Common Components
DESC	The system should support publishing projects with reusable components as NuGet packages (the *PlatformPulumi* & *PlatformPulumiAzure* projects).
RAT	Ensures reusable infrastructure components are available for multiple projects, improving efficiency.
DEP	FR2

ID	FR4
TITLE	Setup IAM Provisioning IaC Projects
DESC	The IaC projects (*EntraId* & *AzureRbac*) must be manually generated, while the post-setup provisioning of required identities and access management should be automated. This enables

	the projects to self-assign necessary identities and access through the Global Administrator, allowing others to work within the IaC projects without requiring additional credentials.
RAT	Ensures an automated, credential-free workflow after the initial setup, enhancing security and usability.
DEP	FR3

ID	FR5
TITLE	Include CI/CD pipelines in projects
DESC	Each project generated by the template suite should have an integrated CI/CD pipeline for automated building, testing, and deployment.
RAT	Standardizes deployments, reduces manual intervention, and ensures consistency across environments.
DEP	FR4

4.2 IAM Administrators

ID	FR6
TITLE	Manage IAM Provisioning IaC Projects
DESC	IAM Administrators must be able to provision Microsoft Entra ID applications with FICs, MIs, Resource Groups, and Azure RBAC role assignments, from the IaC projects (*EntraId* & *AzureRbac*)
RAT	Provides automated and version controlled IAM, improving efficiency, visibility and security of the infrastructure.
DEP	FR5

4.3 Operations & Developer Teams

ID	FR7
TITLE	Provision Microservice Resources
DESC	Operators should be able to provision microservice-specific resources in assigned resource groups, with assigned identities and roles. It should be automated in a CI/CD pipeline using an assigned FIC to gain access.
RAT	Ensures microservices are securely deployed with the correct permissions in an isolated environment.
DEP	FR6

ID	FR8
TITLE	Access Microservice Resources
DESC	Developers should be able to use MIs for secure access to Azure resources within internal workloads.
RAT	Enables authentication without storing sensitive credentials, improving security and compliance.
DEP	FR7

5 Non-Functional Requirements

This section outlines the non-functional requirements that define quality attributes of the system regarding performance, maintainability, usability, and security.

5.1 Performance

ID	QR1
TITLE	Optimized System Performance
DESC	The system components, including infrastructure code and CI/CD pipelines, should be optimized to ensure that deploying resources are performed in under 5 minutes.
RAT	Improves agility and reduces pipeline costs and time consumption of pipeline monitoring.
DEP	None

5.2 Maintainability

ID	QR2
TITLE	Code Readability
DESC	The system should enforce clear, well-documented, and structured code that is easy to read, understand, and extend.
RAT	Facilitates future modifications and reduces complexity for developers.
DEP	None

ID	QR3

TITLE	Reusable Components & Modularization
DESC	The system should be modularized to maximize reuse of resource builders, naming conventions, and other components.
RAT	Ensures consistency across projects and minimizes redundant code.
DEP	None

ID	QR4
TITLE	Testability
DESC	The system should include a test environment, while ensuring configuration consistency with production environments.
RAT	Provides an experimental environment while preventing inconsistencies with production environments.
DEP	QR3

5.3 Usability

ID	QR5
TITLE	Documentation
DESC	The system must include detailed documentation covering setup, usage, and troubleshooting.
RAT	Improves onboarding and reduces support overhead.
DEP	None

5.4 Security

ID	QR6
TITLE	Credential-free Workload Identities
DESC	The system must use credential-free workload identities throughout the entire infrastructure, for Azure resource access in both internal and external workloads.
RAT	Enhances security by eliminating reliance on long-lived credentials, thereby reducing the risk of unauthorized access to resources.
DEP	None

ID	QR7
TITLE	IAM Role Least Privilege Enforcement
DESC	The system should follow least privilege access principles when provisioning identities and roles.
RAT	Reduces security risks by limiting unnecessary access.
DEP	None

6 Documentation

The template suite NuGet package will contain a README with sections on pre-requisites, installation, dependencies between templates and other necessary information. Additionally, each template will contain README files providing information on how to set up and manage the template.

7 Conditions for Delivery

The templates will be consolidated into a single repository and distributed as a NuGet package. Users will install the package locally, integrating it with the .NET template engine[12]. Once installed, the templates can be used to generate customized projects via the `dotnet new` command with configurable parameters.

[12] https://github.com/dotnet/templating

References

[1] S. Ahmadi, "Systematic literature review on cloud computing security: Threats and mitigation strategies," *Journal of Information Security*, vol. 15, pp. 148–167, Mar. 2024.

[2] A. Puchta, F. Böhm, and G. Pernul, "Contributing to current challenges in identity and access management with visual analytics," *Proc. 33rd IFIP WG 11.3 Conf. Data Appl. Secur. Privacy (DBSec 2019)*, vol. 11559, pp. 221-239, Cham, Switzerland, Jul. 2019.

[3] H. Wang, B. Kishiyama, D. Lopez, and J. Yang, "An overview of Infrastructure as Code (IaC) with performance and availability assessment on Google Cloud Platform," *Proc. 2nd Int. Conf. Adv. Comput. Res. (ACR'24)*, pp. 497–514, Mar. 2024

[4] D. Verner, "The development of Infrastructure as Code practices for improving IT infrastructure management efficiency," *Norwegian J. Dev. Int. Sci.*, no. 142, p. 75-78, 2024.

[5] J. Lepiller, R. Piskac, M. Schäf, and M. Santolucito, "Analyzing Infrastructure as Code to prevent intra-update sniping vulnerabilities," *Proc. 27th Int. Conf. Tools Algorithms Constr. Anal. Syst. (TACAS 2021)*, pp. 105–123, Mar.–Apr. 2021. Springer.

Appendix A: UC-1 - Setup Template Suite

Identifier		UC-1: Setup Template Suite	
Purpose		Initialize foundational projects that can be used by Operations and Developers for further development and deployment.	
Priority		High	
Pre-conditions			
Post-conditions		● Baseline projects are set up and integrated with third party actors. ● Baseline IAM resources are provisioned.	
Typical Course of Action			
S#	**Actor**	**Action**	**Result**
1.1	Global Administrator	Performs actions UC-1.1 - UC-1.5	The workload IAM framework was provisioned and configured.

Appendix B: UC-1.1 - Install Template Suite

Identifier	UC-1.1: Install Template Suite
Purpose	Install template suite
Priority	High
Pre-conditions	● .NET SDK installed
Post-conditions	● Complete template suite locally installed and integrated with the .NET template engine, available via the `dotnet new` command.

Typical Course of Action

S#	**Actor**	**Action**	**Result**
1.1	Global Administrator	Installs template suite with: `dotnet new install <PackageName>`	Successful download
1.2	Global Administrator	Verifies installation: `dotnet new list`	A list of available templates are displayed in the console, including the templates from the installed package

Alternate Course of Action

S#	**Actor**	**Action**	**Result / Resolution**
2	Global Administrator	Installs template suite	Unsuccessful download. Refer to .NET CLI logs and template documentation for troubleshooting steps. Retry after necessary fixes.

Appendix C: UC-1.2 - Setup PlatformPulumi Project

Identifier	UC-1.2: Setup PlatformPulumi Project
Purpose	To generate the *PlatformPulumi* project based on a predefined template and publish it as a NuGet package to a GitLab Package Registry.
Priority	High
Pre-conditions	● UC-1.1 Template Suite Installation step completed, shown in Appendix B. ● GitLab Group and a Project within that Group ○ Group has to be added to the Project's CI/CD job token allowlist ○ Need access to create merge requests (MR), run CI/CD pipelines, and merge to the main branch in that Group or Project.
Post-conditions	● A NuGet package with base Pulumi components and conventions is available in the GitLab Group's Package Registry.

Typical Course of Action

S#	Actor	Action	Result
1.1	Global Administrator	Creates a new Git branch from main: `git checkout -b <BranchName>`	A new branch was created
1.2	Global Administrator & Template	Generates project: `dotnet new PlatformPulumi`, with supported parameters	A new project was generated
1.3	Global Administrator	Commits changes and creates an MR in GitLab	An MR was created
1.4	Template (GitLab CI/CD)	Builds, tests, packages, and publishes a pre-release NuGet upon MR creation	Pipeline succeeded and a pre-release NuGet is available in GitLab Package Registry
1.5	Global Administrator	Merges MR to main branch	Changes were merged into main
1.6	Template (GitLab CI/CD)	Builds, tests, packages and publishes as release NuGet on merge to main	Pipeline succeeded and a release NuGet is available in GitLab Package Registry

Alternate Course of Action

S#	Actor	Action	Result / Resolution
2	Global Administrator & Template	Generates project	Unsuccessful generation. Refer to .NET CLI logs and template documentation for troubleshooting steps. Retry after necessary fixes.
3	Template (GitLab CI/CD)	Creates an MR in GitLab or merges MR to main	Pipeline failed. Refer to pipeline logs and template documentation for troubleshooting

			steps. If merging a change to the main branch causes failures, first retry the pipeline in case of transient issues. If the issue is critical, revert the commit and rerun the pipeline to restore stability. Then, open a new merge request with the necessary fixes before reintroducing the changes.

Appendix D: UC-1.3 - Setup PlatformPulumiAzure Project

Identifier		UC-1.3: Setup PlatformPulumiAzure Project	
Purpose		To generate a *PlatformPulumiAzure* project based on a predefined template, publish it as a NuGet package to a GitLab Package Registry.	
Priority		High	
Pre-conditions		● Same pre-conditions as in UC-1.2, shown in Appendix C. ● The *PlatformPulumi* package must be published and available in the GitLab Group's Package Registry.	
Post-conditions		● A NuGet package, with Azure-specific Pulumi components and conventions, is available in the GitLab Group's Package Registry.	
		Typical Course of Action	
S#	**Actor**	**Action**	**Result**
1.1	Global Administrator	Creates a new Git branch from main: `git checkout -b <BranchName>`	A new branch was created
1.2	Global Administrator & Template	Generates project: `dotnet new PlatformPulumiAzure`, with supported parameters	A new project was generated
1.3	Global Administrator	Commits changes and creates an MR in GitLab	An MR was created
1.4	Template (GitLab CI/CD)	Builds, tests, packages, and publishes a pre-release NuGet upon MR creation	Pipeline succeeded and a pre-release NuGet is available in GitLab Package Registry
1.5	Global Administrator	Merges MR to main branch	Changes were merged into main
1.6	Template (GitLab CI/CD)	Builds, tests, packages and publishes as release NuGet on merge to main	Pipeline succeeded and a release NuGet is available in GitLab Package Registry
		Alternate Course of Action	
S#	**Actor**	**Action**	**Result / Resolution**
2	Global Administrator & Template	Generates project	Unsuccessful generation. Refer to .NET CLI logs and template documentation for troubleshooting steps.
3	Template (GitLab CI/CD)	Creates an MR in GitLab or merges MR to main	Pipeline failed. Refer to pipeline logs and template documentation for troubleshooting steps. If merging a change to the main branch causes failures, first retry the pipeline in

			case of transient issues. If the issue is critical, revert the commit and rerun the pipeline to restore stability. Then, open a new merge request with the necessary fixes before reintroducing the changes.

Appendix E: UC-1.4 - Setup EntraId Project

Identifier	UC-1.4: Setup EntraId Project
Purpose	To generate an *EntraId* project from a predefined template while also executing post-actions to configure necessary settings, permissions, and access, ensuring the project is fully set up and operational.
Priority	High
Pre-conditions	● Same pre-conditions as in UC-1.3, shown in Appendix D. ● The *PlatformPulumiAzure* package must be published and available in the GitLab Group's Package Registry. ● An Azure tenant must exist with two subscriptions: Dev and Prod ● Azure CLI must be installed and authenticated locally with a user holding the Global Administrator role ● Pulumi Cloud must be set up ● Pulumi CLI must be installed and authenticated locally with an administrator account ● The GitLab Group must have access to Pulumi
Post-conditions	● A Microsoft Entra ID Application is provisioned for the *EntraId* project, including necessary directory roles and an associated deployment FIC. ● The Pulumi stack configuration is updated programmatically to include the FIC ID as the Client ID, enabling access to Microsoft Entra ID-protected resources in future deployments. ● A Microsoft Entra ID Application with a respective deployment FIC is successfully provisioned for the *AzureRbac* project, which will be generated in a subsequent step.

Typical Course of Action

S#	Actor	Action	Result
1.1	Global Administrator	Creates a new Git branch from main: `git checkout -b <BranchName>`	A new branch was created
1.2	Global Administrator & Template	Creates project: `dotnet new EntraId`, with supported parameters	A new project was generated
1.3	Global Administrator & Template (Azure CLI & Pulumi)	Runs post action script which: ● Authenticates Azure: `az login` ● Authenticates Pulumi: `pulumi login` ● Initiates Pulumi stack: `pulumi stack init self` and `pulumi stack init main` ● Provisions the self stack: `pulumi up -r -s self`	Necessary settings were configured
1.4	Global Administrator	Commits changes and creates an MR in GitLab	An MR was created
1.5	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi preview` on MR creation	Pipeline succeeded. Information about resources to be created, modified, or

			deleted are displayed
1.6	Global Administrator	Merges MR to the main branch	Changes were successfully merged into main.
1.7	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi refresh` and `pulumi up` upon merging to the main branch	Pipeline completed successfully. Existing resources were synchronized to prevent drift, and new resource configurations were deployed. Information about resources that were created, modified, or deleted during the process are displayed.
Alternate Course of Action			
S#	**Actor**	**Action**	**Result / Resolution**
2	Global Administrator & Template	Generates project	Unsuccessful generation. Refer to .NET CLI logs and template documentation for troubleshooting steps.
3	Template (GitLab CI/CD & Pulumi)	Creates an MR in GitLab or merges MR to main	Pipeline failed. Refer to pipeline logs and template documentation for troubleshooting steps. If merging a change to the main branch causes failures, first retry the pipeline in case of transient issues. If the issue is critical, revert the commit and rerun the pipeline to restore stability. Then, open a new merge request with the necessary fixes before reintroducing the changes.

Appendix F: UC-1.5 - Setup AzureRbac Project

Identifier	UC-1.5: Setup AzureRbac Project
Purpose	To generate an *AzureRbac* project from a predefined template while also executing post-actions to configure necessary settings, permissions, and access, ensuring the project is fully set up and operational.
Priority	High
Pre-conditions	● Same pre-conditions as in UC-1.4, shown in Appendix E. ● The *EntraId* project must be created and deployed.
Post-conditions	● The necessary Azure RBAC roles are assigned to the *AzureRbac* Microsoft Entra ID Application created by the *EntraId* project, allowing the *AzureRbac* project to delegate additional roles. ● The Pulumi stack configuration is programmatically updated to include the provided FIC ID as the Client ID, enabling the *AzureRbac* project to access Azure resources in future deployments.

Typical Course of Action

S#	Actor	Action	Result
1.1	Global Administrator	Creates a new Git branch from main: `git checkout -b <BranchName>`	A new branch was created
1.2	Global Administrator & Template	Creates project: `dotnet new EntraId`, with supported parameters	A new project was generated
1.3	Global Administrator & Template (Azure CLI & Pulumi)	Runs post action script which: ● Authenticates Azure: `az login` ● Authenticates Pulumi: `pulumi login` ● Initiates Pulumi stack: `pulumi stack init self` and `pulumi stack init main` ● Provisions the self stack: `pulumi up -r -s self`	Necessary settings were configured
1.4	Global Administrator	Commits changes and creates an MR in GitLab	An MR was created
1.5	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi preview` on MR creation	Pipeline succeeded. Information about resources to be created, modified, or deleted are displayed
1.6	Global Administrator	Merges MR to the main branch	Changes were successfully merged into main.
1.7	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi refresh` and `pulumi up` upon merging to the main branch	Pipeline completed successfully. Existing resources were synchronized to prevent drift, and new resource configurations were deployed. Information about resources that were created, modified, or deleted during the

			process are displayed.
Alternate Course of Action			
S#	**Actor**	**Action**	**Result / Resolution**
2	Global Administrator & Template	Generates project	Unsuccessful generation. Refer to .NET CLI logs and template documentation for troubleshooting steps.
3	Template (GitLab CI/CD & Pulumi)	Creates an MR in GitLab or merges MR to main	Pipeline failed. Refer to pipeline logs and template documentation for troubleshooting steps. If merging a change to the main branch causes failures, first retry the pipeline in case of transient issues. If the issue is critical, revert the commit and rerun the pipeline to restore stability. Then, open a new merge request with the necessary fixes before reintroducing the changes.

Appendix G: UC-2 - Manage IAM

Identifier	UC-2: Manage IAM		
Purpose	To logically group and constrain access to specific applications or microservices, while restricting management to designated administrators.		
Priority	High		
Pre-conditions	● UC-1 (See Appendix A)		
Post-conditions	● Provisions workload IAM resources for use by microservices.		
Typical Course of Action			
S#	**Actor**	**Action**	**Result**
1.1	EntraId Administrator	Performs UC-2.1	Microservice-specific Microsoft Entra ID Applications with directory roles and FICs were provisioned. Pulumi Stack Outputs display necessary details on provisioned resources.
1.2	AzureRbac Administrator	Performs UC-2.2	Microservice-specific Resource Groups and MIs were provisioned. These resources, along with Microsoft Entra ID Applications and FICs were given necessary Azure role assignments. Pulumi Stack Outputs display necessary details on provisioned resources.

Appendix H: UC-2.1 - Managing EntraId Project

Identifier	UC-2.1: Managing EntraId Project		
Purpose	To manage the generated *EntraId* project. Administrators can modify and add resources within the project as needed.		
Priority	High		
Pre-conditions	● The full template suite must be set up ● User access to create MRs, run CI/CD pipelines, and merge to the main branch in the GitLab Group or Project		
Post-conditions	● The desired Microsoft Entra ID Application is deployed and ready for use, with specified directory roles, FICs and other configurations. ● Necessary values for use in other Pulumi stacks are available as outputs via stack references.		
Typical Course of Action			
S#	**Actor**	**Action**	**Result**
1.1	EntraId Administrator	Creates a new Git branch from main: `git checkout -b <BranchName>`	A new branch was created
1.2	EntraId Administrator	Makes necessary code changes to manage resources	Resources were created, modified, or deleted in code
1.3	EntraId Administrator	Commits changes and creates an MR in GitLab	An MR was created
1.4	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi preview` on MR creation	Pipeline succeeded. Information about resources to be created, modified, or deleted are displayed
1.5	EntraId Administrator	Merges MR to the main branch	Changes were successfully merged into main
1.6	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi refresh` and `pulumi up` upon merging to the main branch	Pipeline completed successfully. Existing resources were synchronized to prevent drift, and new resource configurations were deployed. Information about resources that were created, modified, or deleted during the process are displayed.
Alternate Course of Action			
S#	**Actor**	**Action**	**Result / Resolution**
2	Template (GitLab CI/CD & Pulumi)	Creates an MR in GitLab or merges MR to main	Pipeline failed. Refer to pipeline logs and template documentation for troubleshooting steps.

			If merging a change to the main branch causes failures, first retry the pipeline in case of transient issues. If the issue is critical, revert the commit and rerun the pipeline to restore stability. Then, open a new merge request with the necessary fixes before reintroducing the changes.

Appendix I: UC-2.2 - Managing AzureRbac Project

Identifier	UC-2.2: Managing AzureRbac Project		
Purpose	To manage the generated *AzureRbac* project. Administrators can modify and add Azure-related roles within the project as needed.		
Priority	High		
Pre-conditions	● The full template suite must be deployed ● User access to create merge requests (MR), run CI/CD pipelines, and merge to the main branch in that project or group		
Post-conditions	● The desired resource changes are deployed and ready for use, with specified Azure RBAC roles and other configurations. ● Necessary values for use in other Pulumi stacks are available as outputs via stack references.		
Typical Course of Action			
S#	**Actor**	**Action**	**Result**
1.1	AzureRbac Administrator	Creates a new Git branch from main: `git checkout -b <BranchName>`	A new branch was created
1.2	AzureRbac Administrator	Makes necessary code changes to manage resources	Resources were created, modified, or deleted in code
1.3	AzureRbac Administrator	Commits changes and creates an MR in GitLab	An MR was created
1.4	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi preview` on MR creation	Pipeline succeeded. Information about resources to be created, modified, or deleted are displayed.
1.5	AzureRbac Administrator	Merges MR to the main branch	Changes were successfully merged into main
1.6	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi refresh` and `pulumi up` upon merging to the main branch	Pipeline completed successfully. Existing resources were synchronized to prevent drift, and new resource configurations were deployed. Information about resources that were created, modified, or deleted during the process are displayed.
Alternate Course of Action			
S#	**Actor**	**Action**	**Result / Resolution**
2	Template (GitLab CI/CD & Pulumi)	Creates an MR in GitLab or merges MR to main	Pipeline failed. Refer to pipeline logs and template documentation for troubleshooting steps.

			If merging a change to the main branch causes failures, first retry the pipeline in case of transient issues. If it fails again, investigate the cause. If the issue is critical, revert the commit and rerun the pipeline to restore stability. Then, open a new merge request with the necessary fixes before reintroducing the changes.

Appendix J: UC-3 - Microservice Development

Identifier	UC-3: Microservice Development		
Purpose	To provision and access resources dedicated to a specific microservice.		
Priority	High		
Pre-conditions	● UC-2 (See Appendix G)		
Post-conditions	● Azure resources dedicated to the microservice are provisioned ● The workloads within the microservice can access the Azure resources		
Typical Course of Action			
S#	**Actor**	**Action**	**Result**
1.1	Operations Developer	Performs UC-3.1	MIcroservice-specific resources were provisioned by an external CI/CD workload, authenticated with an FIC
1.2	Developer	Performs UC-3.2	Microservice workloads have access via MIs

Appendix K: UC-3.1 - Provisioning of microservice resources

Identifier	UC-3.1: Provisioning of microservice resources
Purpose	To provision microservice resources within the assigned resource group.
Priority	High
Pre-conditions	● An EntraId and AzureRbac admin has created necessary workload identities and a resource group for the specific microservice, with values available via stack references. ● User access to create merge requests (MR), run CI/CD pipelines, and merge to the main branch in that project or group.
Post-conditions	● The desired microservice resources are deployed and ready for use.

Typical Course of Action			
S#	**Actor**	**Action**	**Result**
1.1	Operations Administrator	Creates a new Git branch from main: `git checkout -b <BranchName>`	A new branch was created
1.2	Operations Administrator	Makes necessary code changes to manage resources	Resources were created, modified, or deleted in code
1.3	Operations Administrator	Commits changes and creates an MR in GitLab	An MR was created
1.4	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi preview` on MR creation	Pipeline succeeded. Information about resources to be created, modified, or deleted are displayed.
1.5	Operations Administrator	Merges MR to the main branch	Changes were merged into main
1.6	Template (GitLab CI/CD & Pulumi)	Builds, tests, and runs `pulumi refresh` and `pulumi up` upon merging to the main branch	Pipeline completed successfully. Existing resources were synchronized to prevent drift, and new resource configurations were deployed. Information about resources that were created, modified, or deleted during the process are displayed.
Alternate Course of Action			
S#	**Actor**	**Action**	**Result / Resolution**
2	Template (GitLab CI/CD & Pulumi)	Creates an MR in GitLab or merges MR to main	Pipeline failed. Refer to pipeline logs and template documentation for troubleshooting steps. If merging a change to the main branch

			causes failures, first retry the pipeline in case of transient issues. If it fails again, investigate the cause. If the issue is critical, revert the commit and rerun the pipeline to restore stability. Then, open a new merge request with the necessary fixes before reintroducing the changes.

Appendix L: UC-3.2 - Accessing resources from microservice

Identifier		UC-3.2: Accessing resources from microservice	
Purpose		To access Azure resources within its designated resource group, from within microservice workloads.	
Priority		High	
Pre-conditions		● Operations has provisioned microservice resources. ● User access to create merge requests (MR), run CI/CD pipelines, and merge to the main branch in that project or group.	
Post-conditions		● Microservice workloads have access to necessary Azure resources within its resource group without using a secret.	
Typical Course of Action			
S#	**Actor**	**Action**	**Result**
1.1	Developer	Adds MI to workload configurations	Workloads were configured to use the MI for authorization
1.2	Template (MI)	Authorizes workload	Workload was authenticated and granted access to desired Azure resource
Alternate Course of Action			
S#	**Actor**	**Action**	**Result / Resolution**
2	Template (MI)	Authorizes workload	Authentication failed or the MI had insufficient access to the desired resource. Ensure the MI has the correct RBAC role assignments.